NETMOD Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: December 22, 2019

K. Watsen
Watsen Networks
A. Farrel
Old Dog Consulting
Q. Wu
Huawei Technologies

June 20, 2019

Handling Long Lines in Inclusions in Internet-Drafts and RFCs
draft-ietf-netmod-artwork-folding-05

Abstract

This document defines two strategies for handling long lines in width-bounded text content. One strategy is based on the historic use of a single backslash ('\') character to indicate where line-folding has occurred, with the continuation occurring with the first non-space (' ') character on the next line. The second strategy extends the first strategy by adding a second backslash character to identify where the continuation begins and thereby able to handle cases not supported by the first strategy. Both strategies use a self-describing header enabling automated reconstitution of the original content.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[RFC7994] sets out the requirements for plain-text RFCs and states that each line of an RFC (and hence of an Internet-Draft) must be limited to 72 characters followed by the character sequence that denotes an end-of-line (EOL).

Internet-Drafts and RFCs often include example text or code fragments. Many times the example text or code exceeds the 72 character line-length limit. The `xml2rfc` utility does not attempt to wrap the content of such inclusions, simply issuing a warning whenever lines exceed 69 characters. According to the RFC Editor, there is currently no convention in place for how to handle long lines in such inclusions, other than advising authors to clearly indicate what manipulation has occurred.

This document defines two strategies for handling long lines in width-bounded text content. One strategy is based on the historic use of a single backslash ('\') character to indicate where line-folding has occurred, with the continuation occurring with the first non-space (' ') character on the next line. The second strategy extends the first strategy by adding a second backslash character to identify where the continuation begins and thereby able to handle cases not supported by the first strategy. Both strategies use a self-describing header enabling automated reconstitution of the original content.

The strategies defined in this document work on any text content, but are primarily intended for a structured sequence of lines, such as would be referenced by the <sourcecode> element defined in Section 2.48 of [RFC7991], rather than for two-dimensional imagery, such as would be referenced by the <artwork> element defined in Section 2.5 of [RFC7991].

Note that text files are represented as lines having their first character in column 1, and a line length of N where the last character is in the Nth column and is immediately followed by an end of line character sequence.

2. Applicability Statement

The formats and algorithms defined in this document may be used in any context, whether for IETF documents or in other situations where structured folding is desired.

Within the IETF, this work primarily targets the xml2rfc v3 <sourcecode> element (Section 2.48 of [RFC7991]) and the xml2rfc v2 <artwork> element (Section 2.5 of [RFC7749]) that, for lack of a better option, is currently used for both source code and artwork. This work may be also be used for the xml2rfc v3 <artwork> element (Section 2.5 of [RFC7991]) but, as described in Section 5.1, it is generally not recommended.

3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

4. Goals

4.1. Automated Folding of Long Lines in Text Content

Automated folding of long lines is needed in order to support draft compilations that entail a) validation of source input files (e.g., XML, JSON, ABNF, ASN.1) and/or b) dynamic generation of output, using a tool that doesn't observe line lengths, that is stitched into the final document to be submitted.

Generally, in order for tooling to be able to process input files, the files must be in their original/natural state, which may entail them having some long lines. Thus, these source files need to be modified before inclusion in the document in order to satisfy the line length limits. This modification SHOULD be automated to reduce effort and errors resulting from manual processing.

Similarly, dynamically generated output (e.g., tree diagrams) must also be modified, if necessary, in order for the resulting document to satisfy the line length limits. When needed, this effort again SHOULD be automated to reduce effort and errors resulting from manual processing.

4.2. Automated Reconstitution of the Original Text Content

Automated reconstitution of the exact original text content is needed to support validation of text-based content extracted from documents.

For instance, already YANG [RFC7950] modules are extracted from Internet-Drafts and validated as part of the draft-submission process. Additionally, the desire to validate instance examples (i.e., XML/JSON documents) contained within Internet-Drafts has been discussed ([yang-doctors-thread]).

5. Limitations

5.1. Not Recommended for Graphical Artwork

While the solution presented in this document works on any kind of text-based content, it is most useful on content that represents source code (XML, JSON, etc.) or, more generally, on content that has not been laid out in two dimensions (e.g., diagrams).

Fundamentally, the issue is whether the text content remains readable once folded. Text content that is unpredictable is especially susceptible to looking bad when folded; falling into this category are most UML diagrams, YANG tree diagrams, and ASCII art in general.

It is NOT RECOMMENDED to use the solution presented in this document on graphical artwork.

5.2. Doesn't Work as Well as Format-Specific Options

The solution presented in this document works generically for all text-based content, as it only views content as plain text. However, various formats sometimes have built-in mechanisms that are better suited to prevent long lines.

For instance, both the `pyang` and `yanglint` utilities have the command line option "--tree-line-length" that can be used to indicate a desired maximum line length for when generating tree diagrams [RFC8340].

In another example, some source formats (e.g., YANG [RFC7950]) allow any quoted string to be broken up into substrings separated by a concatenation character (e.g., '+'), any of which can be on a different line.

It is RECOMMENDED that authors do as much as possible within the selected format to avoid long lines.

6. Two Folding Strategies

This document defines two nearly identical strategies for folding text-based content.

The Single Backslash Strategy ('\'): Uses a backslash ('\') character at the end of the line where folding occurs, and assumes that the continuation begins at the character that is not a space character (' ') on the following line.

The Double Backslash Strategy ('\\'): Uses a backslash ('\') character at the end of the line where folding occurs, and assumes that the continuation begins after a second backslash ('\') character on the following line.

6.1. Comparison

The first strategy produces more readable output, however it is significantly more likely to encounter unfoldable input (e.g., there exists a line anywhere in the input ending with a backslash character, or there exists a long line containing only space characters) and, for long lines that can be folded, automation implementations may encounter scenarios that will produce errors without special care.

The second strategy produces less readable output, but is unlikely to encounter unfoldable input, there are no long lines that cannot be folded, and no special care is required for when folding a long line.

6.2. Recommendation

It is RECOMMENDED for implementations to first attempt to fold content using the single backslash strategy and, only in the unlikely event that it cannot fold the input or the folding logic is unable to cope with a contingency occurring on the desired folding column, then fallback to the double backslash strategy.

7. The Single Backslash Strategy ('\')

7.1. Folded Structure

Text content that has been folded as specified by this strategy MUST adhere to the following structure.

7.1.1. Header

The header is two lines long.

The first line is the following 45-character string that MAY be surrounded by any number of printable characters. This first line cannot itself be folded.

NOTE: '\' line wrapping per BCP XX (RFC XXXX)

[Note to RFC Editor: Please replace XX and XXXX with the numbers assigned to this document and delete this note. Please make this change in multiple places in this document.]

The second line is a blank line. This line provides visual separation for readability.

7.1.2. Body

The character encoding is the same as described in Section 2 of [RFC7994], except that, per [RFC7991], tab characters are prohibited.

Lines that have a backslash ('\') occurring as the last character in a line are considered "folded".

Really long lines may be folded multiple times.

7.2. Algorithm

This section describes a process for folding and unfolding long lines when they are encountered in text content.

The steps are complete, but implementations MAY achieve the same result in other ways.

When a larger document contains multiple instances of text content that may need to be folded or unfolded, another process must insert/extract the individual text content instances to/from the larger document prior to utilizing the algorithms described in this section. For example, the `xiax` utility [xiax] does this.

7.2.1. Folding

Determine the desired maximum line length from input to the line-wrapping process, such as from a command line parameter. If no value is explicitly specified, the value "69" SHOULD be used.

Ensure that the desired maximum line length is not less than the minimum header, which is 45 characters. If the desired maximum line length is less than this minimum, exit (this text-based content cannot be folded).

Scan the text content for horizontal tab characters. If any horizontal tab characters appear, either resolve them to space characters or exit, forcing the input provider to convert them to space characters themselves first.

Scan the text content to ensure at least one line exceeds the desired maximum. If no line exceeds the desired maximum, exit (this text content does not need to be folded).

Scan the text content to ensure no existing lines already end with a backslash ('\') character, as this would lead to an ambiguous result. If such a line is found, exit (this text content cannot be folded).

If this text content needs to and can be folded, insert the header described in Section 7.1.1, ensuring that any additional printable characters surrounding the header does not result in a line exceeding the desired maximum.

For each line in the text content, from top-to-bottom, if the line exceeds the desired maximum, then fold the line by:

1. Determine where the fold will occur. This location MUST be before or at the desired maximum column, and MUST NOT be chosen such that the character immediately after the fold is a space (' ') character. If no such location can be found, then exit (this text content cannot be folded)

2. At the location where the fold is to occur, insert a backslash ('\') character followed by the end of line character sequence.

3. On the following line, insert any number of space (' ') characters.

The result of the previous operation is that the next line starts with an arbitrary number of space (' ') characters, followed by the character that was previously occupying the position where the fold occurred.

Continue in this manner until reaching the end of the text content. Note that this algorithm naturally addresses the case where the remainder of a folded line is still longer than the desired maximum, and hence needs to be folded again, ad infinitum.

The process described in this section is illustrated by the "fold_it_1()" function in Appendix A.

7.2.2. Unfolding

Scan the beginning of the text content for the header described in Section 7.1.1. If the header is not present, starting on the first line of the text content, exit (this text contents does not need to be unfolded).

Remove the 2-line header from the text content.

For each line in the text content, from top-to-bottom, if the line has a backslash ('\') character immediately followed by the end of line character sequence, then the line can be unfolded. Remove the backslash ('\') character, the end of line character sequence, and any leading space (' ') characters, which will bring up the next line. Then continue to scan each line in the text content starting with the current line (in case it was multiply folded).

Continue in this manner until reaching the end of the text content.

The process described in this section is illustrated by the "unfold_it_1()" function in Appendix A.

8. The Double Backslash Strategy ('\\')

8.1. Folded Structure

Text content that has been folded as specified by this strategy MUST adhere to the following structure.

8.1.1. Header

The header is two lines long.

The first line is the following 46-character string that MAY be surrounded by any number of printable characters. This first line cannot itself be folded.

NOTE: '\\' line wrapping per BCP XX (RFC XXXX)

[Note to RFC Editor: Please replace XX and XXXX with the numbers assigned to this document and delete this note. Please make this change in multiple places in this document.]

The second line is a blank line. This line provides visual separation for readability.

8.1.2. Body

The character encoding is the same as described in Section 2 of [RFC7994], except that, per [RFC7991], tab characters are prohibited.

Lines that have a backslash ('\') occurring as the last character in a line immediately followed by the end of line character sequence, when the subsequent line starts with a backslash ('\') as the first non-space (' ') character, are considered "folded".

Really long lines may be folded multiple times.

8.2. Algorithm

This section describes a process for folding and unfolding long lines when they are encountered in text content.

The steps are complete, but implementations MAY achieve the same result in other ways.

When a larger document contains multiple instances of text content that may need to be folded or unfolded, another process must insert/extract the individual text content instances to/from the larger document prior to utilizing the algorithms described in this section. For example, the `xiax` utility [xiax] does this.

8.2.1. Folding

Determine the desired maximum line length from input to the line-wrapping process, such as from a command line parameter. If no value is explicitly specified, the value "69" SHOULD be used.

Ensure that the desired maximum line length is not less than the minimum header, which is 46 characters. If the desired maximum line length is less than this minimum, exit (this text-based content cannot be folded).

Scan the text content for horizontal tab characters. If any horizontal tab characters appear, either resolve them to space characters or exit, forcing the input provider to convert them to space characters themselves first.

Scan the text content to see if any line exceeds the desired maximum. If no line exceeds the desired maximum, exit (this text content does not need to be folded).

Scan the text content to ensure no existing lines already end with a backslash ('\') character while the subsequent line starts with a

backslash ('\') character as the first non-space (' ') character, as this could lead to an ambiguous result. If such a line is found, and its width is less than the desired maximum, then it SHOULD be flagged for forced folding (folding even though unnecessary). If the folding implementation doesn't support forced foldings, it MUST exit.

If this text content needs to and can be folded, insert the header described in Section 8.1.1, ensuring that any additional printable characters surrounding the header does not result in a line exceeding the desired maximum.

For each line in the text content, from top-to-bottom, if the line exceeds the desired maximum, or requires a forced folding, then fold the line by:

1. Determine where the fold will occur. This location MUST be before or at the desired maximum column.

2. At the location where the fold is to occur, insert a first backslash ('\') character followed by the end of line character sequence.

3. On the following line, insert any number of space (' ') characters followed by a second backslash ('\') character.

The result of the previous operation is that the next line starts with an arbitrary number of space (' ') characters, followed by a backslash ('\') character, immediately followed by the character that was previously occupying the position where the fold occurred.

Continue in this manner until reaching the end of the text content. Note that this algorithm naturally addresses the case where the remainder of a folded line is still longer than the desired maximum, and hence needs to be folded again, ad infinitum.

The process described in this section is illustrated by the "fold_it_2()" function in Appendix A.

8.2.2. Unfolding

Scan the beginning of the text content for the header described in Section 8.1.1. If the header is not present, starting on the first line of the text content, exit (this text content does not need to be unfolded).

Remove the 2-line header from the text content.

For each line in the text content, from top-to-bottom, if the line has a backslash ('\') character immediately followed by the end of line character sequence, and if the next line has a backslash ('\') character as the first non-space (' ') character, then the lines can be unfolded. Remove the first backslash ('\') character, the end of line character sequence, any leading space (' ') characters, and the second backslash ('\') character, which will bring up the next line. Then continue to scan each line in the text content starting with the current line (in case it was multiply folded).

Continue in this manner until reaching the end of the text content.

The process described in this section is illustrated by the "unfold_it_2()" function in Appendix A.

9. Examples

The following self-documenting examples illustrate folded text-based content.

The source text content cannot be presented here, as it would again be folded. Alas, only the results can be provided.

9.1. Example Showing Boundary Conditions

This example illustrates boundary condition. The input contains seven lines, each line one character longer than the previous line. Numbers for counting purposes. The default desired maximum column value "69" is used.

9.1.1. Using '\'

```
=========== NOTE: '\' line wrapping per BCP XX (RFC XXXX) ===========

123456789012345678901234567890123456789012345678901234567890123456
123456789012345678901234567890123456789012345678901234567890123456789012345678
123456789012345678901234567890123456789012345678901234567890123456789012345678
123456789012345678901234567890123456789012345678901234567890123456789
123456789012345678901234567890123456789012345678901234567890123456789012345678\
90
123456789012345678901234567890123456789012345678901234567890123456789012345678\
901
123456789012345678901234567890123456789012345678901234567890123456789012345678\
9012
```

9.1.2. Using '\\'

```
========== NOTE: '\\' line wrapping per BCP XX (RFC XXXX) ===========

123456789012345678901234567890123456789012345678901234567890123456
1234567890123456789012345678901234567890123456789012345678901234567
12345678901234567890123456789012345678901234567890123456789012345678
123456789012345678901234567890123456789012345678901234567890123456789
12345678901234567890123456789012345678901234567890123456789012345678\
\90
12345678901234567890123456789012345678901234567890123456789012345678\
\901
12345678901234567890123456789012345678901234567890123456789012345678\
\9012
```

9.2. Example Showing Multiple Wraps of a Single Line

This example illustrates what happens when very long line needs to be folded multiple times. The input contains one line containing 280 characters. Numbers for counting purposes. The default desired maximum column value "69" is used.

9.2.1. Using '\'

```
=========== NOTE: '\' line wrapping per BCP XX (RFC XXXX) ===========

12345678901234567890123456789012345678901234567890123456789012345678\
90123456789012345678901234567890123456789012345678901234567890123456\
78901234567890123456789012345678901234567890123456789012345678901234\
56789012345678901234567890123456789012345678901234567890123456789012\
34567890
```

9.2.2. Using '\\'

```
========== NOTE: '\\' line wrapping per BCP XX (RFC XXXX) ===========

12345678901234567890123456789012345678901234567890123456789012345678\
\9012345678901234567890123456789012345678901234567890123456789012345\
\6789012345678901234567890123456789012345678901234567890123456789012\
\3456789012345678901234567890123456789012345678901234567890123456789\
\01234567890
```

9.3. Example Showing Smart Folding

This example illustrates how readability can be improved via "smart" folding, whereby folding occurs at format-specific locations and format-specific indentations are used.

The text content was manually folded, since the script in the appendix does not implement smart folding.

Note that the header is surrounded by different printable characters then shown in the script-generated examples.

9.3.1. Using '\'

[NOTE: '\' line wrapping per BCP XX (RFC XXXX)]

```
<yang-library
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">

  <module-set>
    <name>config-modules</name>
    <module>
      <name>ietf-interfaces</name>
      <revision>2018-02-20</revision>
      <namespace>\
        urn:ietf:params:xml:ns:yang:ietf-interfaces\
      </namespace>
    </module>
    ...
  </module-set>
  ...
</yang-library>
```

Below is the equivalent to the above, but it was folded using the script in the appendix.

```
=========== NOTE: '\' line wrapping per BCP XX (RFC XXXX) ===========

<yang-library
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">

  <module-set>
    <name>config-modules</name>
    <module>
      <name>ietf-interfaces</name>
      <revision>2018-02-20</revision>
      <namespace>urn:ietf:params:xml:ns:yang:ietf-interfaces</namesp\
ace>
    </module>
    ...
  </module-set>
  ...
</yang-library>
```

9.3.2. Using '\\'

```
[NOTE: '\\' line wrapping per BCP XX (RFC XXXX)]

<yang-library
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">

  <module-set>
    <name>config-modules</name>
    <module>
      <name>ietf-interfaces</name>
      <revision>2018-02-20</revision>
      <namespace>\
        \urn:ietf:params:xml:ns:yang:ietf-interfaces\
      \</namespace>
    </module>
    ...
  </module-set>
  ...
</yang-library>
```

Below is the equivalent to the above, but it was folded using the script in the appendix.

```
========== NOTE: '\\' line wrapping per BCP XX (RFC XXXX) ===========

<yang-library
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">

  <module-set>
    <name>config-modules</name>
    <module>
      <name>ietf-interfaces</name>
      <revision>2018-02-20</revision>
      <namespace>urn:ietf:params:xml:ns:yang:ietf-interfaces</namesp\
\ace>
    </module>
    ...
  </module-set>
  ...
</yang-library>
```

10. Security Considerations

This BCP has no Security Considerations.

11. IANA Considerations

This BCP has no IANA Considerations.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

12.2. Informative References

[RFC7749] Reschke, J., "The "xml2rfc" Version 2 Vocabulary", RFC 7749, DOI 10.17487/RFC7749, February 2016, <https://www.rfc-editor.org/info/rfc7749>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7991] Hoffman, P., "The "xml2rfc" Version 3 Vocabulary", RFC 7991, DOI 10.17487/RFC7991, December 2016, <https://www.rfc-editor.org/info/rfc7991>.

[RFC7994] Flanagan, H., "Requirements for Plain-Text RFCs", RFC 7994, DOI 10.17487/RFC7994, December 2016, <https://www.rfc-editor.org/info/rfc7994>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[xiax] "The `xiax` Python Package", <https://pypi.org/project/xiax/>.

[yang-doctors-thread] "[yang-doctors] automating yang doctor reviews", <https://mailarchive.ietf.org/arch/msg/yang-doctors/DCfBqgfZPAD7afzeDFlQ1Xm2X3g>.

Appendix A. POSIX Shell Script

This non-normative appendix section includes a shell script that can both fold and unfold text content using both the single and double backslash strategies described in Section 7 and Section 8 respectively.

This script is intended to be applied to a single text content instance. If it is desired to fold or unfold test content instances within a larger document (e.g., an Internet draft or RFC), then another tool must be used to extract the content from the larger document before utilizing this script.

For readability purposes, this script forces the minimally supported line length to be eight characters longer than the raw header text defined in Section 7.1.1 and Section 8.1.1 so as to ensure that the header can be wrapped by a space (' ') character and three equal ('=') characters on each side of the raw header text.

This script does not implement the "forced folding" logic described in Section 8.2.1. In such cases the script will exit with the message:

```
Error: infile has a line ending with a '\\' character
followed by a '\\' character as the first non-space
character on the next line.  This file cannot be folded.
```

Shell-level end-of-line backslash ('\') characters have been purposely added to the script so as to ensure that the script is itself not folded in this document, thus simplify the ability to copy/paste the script for local use. As should be evident by the lack of the mandatory header described in Section 7.1.1, these backslashes do not designate a folded line, such as described in Section 7.

<CODE BEGINS>

```
#!/bin/bash --posix   # must be `bash` (not `sh`)

print_usage() {
  echo
  echo "Folds the text file, only if needed, at the specified"
  echo "column, according to BCP XX."
  echo
  echo "Usage: $0 [-s <strategy>] [-c <col>] [-r] -i <infile>"
  echo "                                            -o <outfile>"
  echo
  echo "  -s: strategy to use, '1' or '2' (default: try 1, else 2)"
```

```
    echo "  -c: column to fold on (default: 69)"
    echo "  -r: reverses the operation"
    echo "  -i: the input filename"
    echo "  -o: the output filename"
    echo "  -d: show debug messages"
    echo "  -q: quiet (suppress error messages)"
    echo "  -h: show this message"
    echo
    echo "Exit status code: zero on success, non-zero otherwise."
    echo
  }

  # global vars, do not edit
  strategy=0 # auto
  debug=0
  quiet=0
  reversed=0
  infile=""
  outfile=""
  maxcol=69  # default, may be overridden by param
  hdr_txt_1="NOTE: '\\' line wrapping per BCP XX (RFC XXXX)"
  hdr_txt_2="NOTE: '\\\\' line wrapping per BCP XX (RFC XXXX)"
  equal_chars="=============================================="
  space_chars="                                              "
  temp_dir=""

  fold_it_1() {
    # ensure input file doesn't contain the fold-sequence already
    pcregrep -M  "\\\\\n" $infile >> /dev/null 2>&1
    if [[ $? -eq 0 ]]; then
      if [[ $quiet -eq 0 ]]; then
        echo
        echo "Error: infile $infile has a line ending with a '\\'"
        echo "character. This file cannot be folded using the '\\'"
        echo "strategy."
        echo
      fi
      return 1
    fi

    # stash some vars
    testcol=`expr "$maxcol" + 1`
    foldcol=`expr "$maxcol" - 1` # for the inserted '\' char

    # ensure input file doesn't contain whitespace on the fold column
    grep "^.\{$foldcol\} " $infile >> /dev/null 2>&1
    if [[ $? -eq 0 ]]; then
      if [[ $quiet -eq 0 ]]; then
```

```
        echo
        echo "Error: infile has a space character occuring on the"
        echo "folding column. This file cannot be folded using the"
        echo "'\\' strategy."
        echo
      fi
      return 1
    fi

    # center header text
    length=`expr ${#hdr_txt_1} + 2`
    left_sp=`expr \( "$maxcol" - "$length" \) / 2`
    right_sp=`expr "$maxcol" - "$length" - "$left_sp"`
    header=`printf "%.*s %s %.*s" "$left_sp" "$equal_chars"\
                     "$hdr_txt_1" "$right_sp" "$equal_chars"`

    # generate outfile
    echo "$header" > $outfile
    echo "" >> $outfile
    gsed "/.\{$testcol\}/s/\(.\{$foldcol\}\)/\1\\\\\n/g"\
            < $infile >> $outfile

    return 0
  }

  fold_it_2() {
    if [ "$temp_dir" == "" ]; then
      temp_dir=`mktemp -d`
    fi

    # ensure input file doesn't contain the fold-sequence already
    pcregrep -M  "\\\\\n[\ ]*\\\\" $infile >> /dev/null 2>&1
    if [[ $? -eq 0 ]]; then
      if [[ $quiet -eq 0 ]]; then
        echo
        echo "Error: infile has a line ending with a '\\' character"
        echo "followed by a '\\' character as the first non-space"
        echo "character on the next line.  This file cannot be folded"
        echo "using the '\\\\' strategy."
        echo
      fi
      return 1
    fi

    # center header text
    length=`expr ${#hdr_txt_2} + 2`
    left_sp=`expr \( "$maxcol" - "$length" \) / 2`
    right_sp=`expr "$maxcol" - "$length" - "$left_sp"`
```

```
    header=`printf "%.*s %s %.*s" "$left_sp" "$equal_chars"\
                     "$hdr_txt_2" "$right_sp" "$equal_chars"`

    # fold using recursive passes ('g' used in fold_it_1 didn't work)
    if [ -z "$1" ]; then
      # init recursive env
      cp $infile $temp_dir/wip
    fi
    testcol=`expr "$maxcol" + 1`
    foldcol=`expr "$maxcol" - 1` # for the inserted '\' char
    gsed "/.\{$testcol\}/s/\(.\{$foldcol\}\)/\1\\\\\n\\\\/" \
      < $temp_dir/wip >> $temp_dir/wip2
    diff $temp_dir/wip $temp_dir/wip2 > /dev/null 2>&1
    if [ $? -eq 1 ]; then
      mv $temp_dir/wip2 $temp_dir/wip
      fold_it_2 "recursing"
    else
      echo "$header" > $outfile
      echo "" >> $outfile
      cat $temp_dir/wip2 >> $outfile
      rm -rf $temp_dir
    fi
    return 0
  }

  fold_it() {
    # ensure input file doesn't contain a TAB
    grep $'\t' $infile >> /dev/null 2>&1
    if [[ $? -eq 0 ]]; then
      if [[ $quiet -eq 0 ]]; then
        echo
        echo "Error: infile contains a TAB character, which is"
        echo "not allowed."
        echo
      fi
      return 1
    fi

    # check if file needs folding
    testcol=`expr "$maxcol" + 1`
    grep ".\{$testcol\}" $infile >> /dev/null 2>&1
    if [ $? -ne 0 ]; then
      if [[ $debug -eq 1 ]]; then
        echo "nothing to do"
      fi
      cp $infile $outfile
      return -1
    fi
```

```
    if [[ $strategy -eq 1 ]]; then
      fold_it_1
      return $?
    fi
    if [[ $strategy -eq 2 ]]; then
      fold_it_2
      return $?
    fi
    quiet_sav=$quite
    quiet=1
    fold_it_1
    result=$?
    quiet=$quiet_sav
    if [[ $result -ne 0 ]]; then
      if [[ $debug -eq 1 ]]; then
        echo "Folding strategy 1 didn't succeed, trying strategy 2..."
      fi
      fold_it_2
      return $?
    fi
    return 0
  }

  unfold_it_1() {
    temp_dir=`mktemp -d`

    # output all but the first two lines (the header) to wip file
    awk "NR>2" $infile > $temp_dir/wip

    # unfold wip file
    gsed ":x; /.*\\\\$/N; s/\\\\\n[ ]*//; tx" $temp_dir/wip > $outfile

    # clean up and return
    rm -rf $temp_dir
    return 0
  }

  unfold_it_2() {
    temp_dir=`mktemp -d`

    # output all but the first two lines (the header) to wip file
    awk "NR>2" $infile > $temp_dir/wip

    # unfold wip file
    gsed ":x; /.*\\\\$/N; s/\\\\\n[ ]*\\\\//; tx" $temp_dir/wip \
      > $outfile

    # clean up and return
```

```
    rm -rf $temp_dir
    return 0
  }

  unfold_it() {
    # check if file needs unfolding
    line=`head -n 1 $infile`
    result=`echo $line | fgrep "$hdr_txt_1"`
    if [ $? -eq 0 ]; then
      unfold_it_1
      return $?
    fi
    result=`echo $line | fgrep "$hdr_txt_2"`
    if [ $? -eq 0 ]; then
      unfold_it_2
      return $?
    fi
    if [[ $debug -eq 1 ]]; then
      echo "nothing to do"
    fi
    cp $infile $outfile
    return -1
  }

  process_input() {
    while [ "$1" != "" ]; do
      if [ "$1" == "-h" -o "$1" == "--help" ]; then
        print_usage
        exit 1
      fi
      if [ "$1" == "-d" ]; then
        debug=1
      fi
      if [ "$1" == "-q" ]; then
        quiet=1
      fi
      if [ "$1" == "-s" ]; then
        strategy="$2"
        shift
      fi
      if [ "$1" == "-c" ]; then
        maxcol="$2"
        shift
      fi
      if [ "$1" == "-r" ]; then
        reversed=1
      fi
      if [ "$1" == "-i" ]; then
```

```
        infile="$2"
        shift
      fi
      if [ "$1" == "-o" ]; then
        outfile="$2"
        shift
      fi
      shift
    done

    if [[ -z "$infile" ]]; then
      if [[ $quiet -eq 0 ]]; then
        echo
        echo "Error: infile parameter missing (use -h for help)"
        echo
      fi
      exit 1
    fi

    if [[ -z "$outfile" ]]; then
      if [[ $quiet -eq 0 ]]; then
        echo
        echo "Error: outfile parameter missing (use -h for help)"
        echo
        exit 1
      fi
    fi

    if [[ ! -f "$infile" ]]; then
      if [[ $quiet -eq 0 ]]; then
        echo
        echo "Error: specified file \"$infile\" is does not exist."
        echo
        exit 1
      fi
    fi

    if [[ $strategy -eq 2 ]]; then
      min_supported=`expr ${#hdr_txt_2} + 8`
    else
      min_supported=`expr ${#hdr_txt_1} + 8`
    fi
    if [[ $maxcol -lt $min_supported ]]; then
      if [[ $quiet -eq 0 ]]; then
        echo
        echo "Error: the folding column cannot be less than"
        echo "$min_supported."
        echo
```

```
      fi
      exit 1
    fi

    # this is only because the code otherwise runs out of equal_chars
    max_supported=`expr ${#equal_chars} + 1 + ${#hdr_txt_1} + 1\
         + ${#equal_chars}`
    if [[ $maxcol -gt $max_supported ]]; then
      if [[ $quiet -eq 0 ]]; then
        echo
        echo "Error: the folding column cannot be more than"
        echo "$max_supported."
        echo
      fi
      exit 1
    fi
  }

  main() {
    if [ "$#" == "0" ]; then
       print_usage
       exit 1
    fi

    process_input $@

    if [[ $reversed -eq 0 ]]; then
      fold_it
      code=$?
    else
      unfold_it
      code=$?
    fi
    exit $code
  }

  main "$@"

  <CODE ENDS>
```

Acknowledgements

The authors thank the following folks for their various contributions (sorted by first name): Benoit Claise, Gianmarco Bruno, Italo Busi, Joel Jaeggli, Jonathan Hansford, Lou Berger, Martin Bjorklund, and Rob Wilton.

The authors additionally thank the RFC Editor for confirming that there is no set convention today for handling long lines in artwork/sourcecode inclusions.

Authors' Addresses

Kent Watsen
Watsen Networks

EMail: kent+ietf@watsen.net

Adrian Farrel
Old Dog Consulting

EMail: adrian@olddog.co.uk

Qin Wu
Huawei Technologies

EMail: bill.wu@huawei.com